Probability-1

Probability is the one of the most important mathematical concept used everywhere including competitive programming. They can be used in various types of questions i.e. from really obvious to really tough questions.

Let's explore different kind of questions possible on probability.

Die Roll (Extremely Basic):

Question link: Link
Tutorial: We know that to win she needs a number greater than or equal to the largest of two numbers obtained by the other two. This she number of possible output in which she wins is : $6 - max(a, b) + 1$

We know that the probability of win = $\frac{Possible\ outputs\ to\ win}{Total\ possible\ outputs}$

But remember that in the output section it is mentioned that in sample inputs, output is $\frac{1}{2}$ thus we need to make $numerator$ and $denominator$ to be co-primes.

Thus we need to divide the resultant $numerator$ and $denominator$ by their gcd.

Thus solution becomes

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    int a,b;
    cin >> a >> b;
    int numerator = 6 - max(a,b) + 1;
    int denominator = 6;
    int g = __gcd(numerator,denominator);
    cout << numerator/g << "/" << denominator/g;
}
```

Dreamoon and Wifi (Basic because of constraints):

Question link: Link
This question can really be a headache for people with poor math skills (like me) but constraint that the input string as no longer than size of 10 makes it a question to be solved in first 10 minute of contest. Thus making it a perfect div2B question.
Now to solve this question we just need this formula:

$$Probability\ of\ an\ event = \frac{Number\ of\ output\ of\ event}{Total\ possible\ outputs}$$

Approach

Since $n \leq 10$ thus even $O(2^n)$ algorithm will work here.

Thus for everyy $'?'$ sign we can recursively call our function.

```
$include<bits/stdc++.h>
using namespace std;
int possibleOutput = 0;
int eventOutput = 0;
int c = 0;
void calPos(string a, int i, int des, int pos){
    if(i == a.length()){
        if(pos == drazilDestination){
            eventOutput++;
        }
        possibleOutput++;
        return;
    }
    if(a[i] == '?'){
        calPos(a,i+1,drazilDestination,pos+1);
        calPos(a,i+1,drazilDestination,pos-1);
        return ;
    }
    if(a[i] == '-'){
        calPos(a,i+1,drazilDestination,pos-1);
        return ;
    }
    if(a[i] == '+'){
        calPos(a,i+1,drazilDestination,pos+1);
        return ;
    }

}
int main(){
    int drazilDestination = 0;
    string a;
    cin >> a;
    for(int i = 0; i < a.length(); ++i){
        if(a[i] == '-')drazilDestination--;
        else drazilDestination++;
    }
    cin >> a;
    calPos(a,0,drazilDestination,0);
    printf("%.12f",(double(ans)/double(c)));
    return 0;
}
```

Archer (Really easy but need to solved mathematically first):

Question link: Link

In this question, lets work out different scenarios

Let $p = \frac{a}{b}\ and\ q = \frac{c}{d}\ and\ s = (1-p)(1-q)$

SmallR hits on $n^{th} turn$	Probability of Event
P(1)	p
P(2)	$(1-p)(1-q)p = sp$
P(3)	$((1-p)(1-q))^2 p = s^2 p$
P(4)	$((1-p)(1-q))^3 p = s^3 p$
P(n+1)	$((1-p)(1-q))^n p = s^n p$

Thus collective probability will be

$$P(smallR\ winning) = P(1) + P(2) + P(3) + ... + P(n+1)$$

For $n \to \infty$ we get

$$P(winning) = p(s + s^2 + s^3 + ...)$$

Now we know that $p \leq 1\ and\ q \leq 1$, that makes $(1-p) \leq 1\ and\ (1-q) \leq 1$

And thus $(1-p)(1-q) \leq 1$

Thus $P(winning)$ is a sum of Geometric Progression with ratio less than 1

Thus $P(winning) = \frac{p}{1-s}$

Now solving

$$s = (1-p)(1-q) = (1-\frac{a}{b})(1-\frac{c}{d}) = \frac{(b-a)(d-c)}{bd} = \frac{e}{f}(say)$$

$$Thus\ P(winning) = \frac{p}{1-\frac{e}{f}} = \frac{pf}{f-e} = \frac{abd}{b(bd-((b-a)(d-c)))}$$

Now all we have to do is output the following equation

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    double a,b,c,d;
    cin >> a >>  b >> c >> d;
    double ans = (a*b*d)/(b*((b*d)-(b-a)*(d-c)));
    printf("%.10f",ans);
    return 0;
}
```

Little Pony and Expected Values (Good question and needs understandin of expected values):

Question link: Link

This question is very simple and straight-forward but we need to work out mathematics to solve problem of this kind. I learned this concept from video guides of Errichto, you can check his other videos as well, he is an awesome coder.

Let us work out the mathematics behind this problem of this kind.
Consider a dice of 6 face thrown twice and we have to work out the expected value of the bigger of the two scores.

We get this result table

\	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	2	3	4	5	6
3	3	3	3	4	5	6
4	4	4	4	4	5	6
5	5	5	5	5	5	6
6	6	6	6	6	6	6

Here one can observe that any number n is at lower-left boundary of its row and column from first one.

For example for number 3, we have this

\	1	2	3
1	1	2	3
2	2	2	3

\	1	2	3
3	3	3	3

Thus 3 occurs $3^2 - 2^2$ times, and hence occurance of any number n is finally reduced to $n^2 - (n-1)^2$

Since probability of each event happening is $(\frac{1}{6})^2$

That makes probability of n to be $P(n) = (\frac{n}{6})^2 - (\frac{(n-1)}{6})^2$

Thus Expected Value becomes

$$E.V. = \sum_{i=1}^{faces} i \times P(i) = i \times ((\frac{i}{6})^k - (\frac{(i-1)}{k})^2)$$

where k stands for number of times it has been thrown.

Now we got a nice and simple formula as the solution to the problem, we can easily code it out.

But remember it is always better to use binary exponentiation to calculate power as it works in $O(log(n))$ complexity

```
#include<bits/stdc++.h>
using namespace std;
double power(double a, int n){
    if(n == 1)return a;
    if(n == 0)return 1;
    double p = power(a,n/2);
    p*=p;
    if(n%2)p*=a;
    return p;
}

int main(){
    int n;
    double m;
    cin >> m >> n;
    double ans = 0;
    for(int i = 1; i <= m; ++i){
        double p = double(i)*(power(i/m,n)-power((i-1)/m,n));
        ans+=p;
    }
    printf("%.10f",ans);
    return 0;
}
```

Expecting Trouble (Basic Question for expected values):

Note: This Question can only be submitted in programming language ADA, we are just taking this question to learn about a basic concept related to the expected values, that is their linear property
Question link: Link

In this question, we need to find the expected value of Friday being bad.

Let us consider this string

010101
Here Length of string $= 6$
and we can calculate expected value as

Here Expected Value of day 0 being bad is 0
but that of day 1 being bad is $\frac{1}{6}$ and so on.
Thus net expected value isf

$$E(n) = \sum_{i=1}^{n} E(i)$$

and thus answer for the above query is $\frac{1}{2}$

But for the given sample string $01?10??10000$ with probability of $?$ being bad friday is 0.5,

Thus we can do it with this
$E(1) = \frac{1}{12}$, $E(0) = 0$, and $E(?) = \frac{1}{12} \times 0.5 = \frac{1}{24}$

$$E(n) = \sum_{i=1}^{n} E(i) = 3 \times E(1) + 3 \times E(?) = 3(\frac{1}{12} + \frac{1}{24}) = 3 \times (\frac{3}{24}) = \frac{3}{8} = 0.375$$

Thus we can easily code it from here on.

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    string a;
    double p,ans=0;
    cin >> a >> p;
    double len = double(a.size());
    for(int i = 0; i < a.size(); ++i){
        if(a[i] == '1'){
            ans+=(1/len);
        }else if(a[i] == '?'){
            ans+=(p/len);
        }
    }
    printf("%.10f",ans);
    return 0;
}
```

Wet Shark and Flowers (Fairly simple but math we might have forgotten):

Question link: Link

Now question is again based on linearity of expected values that we might not need to take all case of all pairs at once but rather take one pair at a time and solve it easily.

Now for all pairs x and $(x+1)\%N$ we just need to calculate the probability of this pair getting $\$\,1000$ from wet shark and add it to answer.

So answer becomes

$$\sum_{i=0}^{n} P(pair\ i\ and (i+1)\%n)$$

Now lets consider any pair i and $(i+1)$, lets say second one to be j.

Numbers to select in i = $r_i - l_i + 1$
Numbers divisible by p in i = $\frac{r_i}{p} - \frac{l_i - 1}{p}$
Probability of selecting a number divisible by p in i = $\frac{\frac{r_i}{p} - \frac{l_i - 1}{p}}{r_i - l_i + 1}$

And same goes for j, but now there are 3 possible situations that product of both of them becomes multiple of p and those are

1. Number choosen in i is multiple of p but that in j is not
2. Number choosen in j is multiple of p but that in i is not
3. Number choosen in i and j are both multiple of p

But there is only one case for which product would not be divisble by p, that is that both numbers are not divisible by p

Hence we can use this formula

$$P(divisible\ by\ p) = 1 - P(Not\ divisible\ by\ p)$$

Thus now we can calculate the result easily

```
int main(){
    ll n,p;
    cin >> n >> p;
    V<double> r(n);
    V<double> l(n);
    loop(i,0,n)cin >> l[i] >> r[i];
    double ans = 0;
    for(int i = 0; i+1 < n; ++i){
        double a1 = long(r[i]/p) - long((l[i]-1)/p);
        a1 = r[i] - l[i] + 1 - a1;
        a1/=(r[i]-l[i] + 1);
        double a2 = long(r[i+1]/p) - long((l[i+1]-1)/p);
        a2 = r[i+1] - l[i+1] + 1 - a2;
        a2/=(r[i+1]-l[i+1] + 1);
        ans+=(1-(a1*a2));
    }
    double a1 = long(r[0]/p) - long((l[0]-1)/p);
    a1 = r[0] - l[0] + 1 - a1;
    a1/=(r[0]-l[0] + 1);
    double a2 = long(r[n-1]/p) - long((l[n-1]-1)/p);
    a2 = r[n-1] - l[n-1] + 1 - a2;
    a2/=(r[n-1]-l[n-1] + 1);
    ans+=(1-(a1*a2));
    printf("%.10f",ans*2000);
    return 0;
}
```

Basket Ball Team (Simple but confusion on evaluating expressions):

Question link: Link

This is again using the concept of previous question. Rather than adding the cases of different selection we find probability that no one of the teammate is from same department and subtract it from 1

Total number of players required $= n$
Total number of players present $= s$
Players in Herr Wafa's Department $= a_h$
Possible ways of selecting teams if Herr Wafa is definitely selected $= {}^{s-1}C_{n-1}$
Possible ways of selecting teams if Herr Wafa is definitely selected and no other player from department h is selected$= {}^{s-a_h-1}C_{n-1}$

Probability that no other teammate will be from same department $= \frac{^{s-a_h-1}C_{n-1}}{^{s-1}C_{n-1}}$

This results to this equation $= \frac{(s-a_h-1)!\ (s-1-(n-1))!}{(s-a_h-1-(n-1))!\ (s-1)!}$

There is always trouble to solve factorials but here we can find a sweet and simple pattern, that on further solving that equation we get

$$\frac{(s-a_h-1)(s-a_h-2)...(s-a_h-(n-2))}{(s-1)(s-2)(s-3)...(s-1-(n-2))}$$

This can be written as

$$\prod_{i=0}^{n-2} \frac{s-a_h-1-i}{s-1-i}$$

Thus now it can be solved in $O(n)$ time complexity

```
int main(){
    int n,m,h;
    cin >> n >> m >> h;
    h--;
    V<int> arr(m);
    int s = 0;
    loop(i,0,m){cin >> arr[i];s+=arr[i];}
    if(s < n){
        cout << "-1.0000000000";
        return 0;
    }
    s-=arr[h];
    s++;
    if(s < n){
        cout << "1.0000000000";
        return 0;
    }
    s--;
    s+=arr[h];
    double ans = 1;
    for(int i = 0; i+1 < n; ++i){
        ans*=(double(s-arr[h]-i)/double(s-1-i));
        // cout << double(s-arr[h]-i) << " " << double(s-1-i)<< " " << (double(s-arr[h]
    }
    printf("%.10f",1-ans);
    return 0;
}
```

Andrey and the Problem (All Math):

Question link: Link

A good solution is provided here as well: Editorial

I took up this problem because it uses good mathematics and it took some time for me to understand the math provided in editorial.

Let us say that we select t people initially and success is defined that only 1 of these t people will create a problem.

Let $f(i)$ be probability that i^{th} person creates the problem and no other person create a problem.

So, $f(i) = p_i \prod_{j=1}^{t}(1-p_j)$ ... $(i \neq j)$
But we can write this equation in following way as well
$f(i) = \frac{p_i}{1-p_i} \prod_{j=1}^{t}(1-p_j)$

which makes $\prod_{j=1}^{t}(1-p_j)$ this a constant value and let's call it P

Now out of t people there are multiple cases those are that either first person creates the question and everyone else fails or second one do so and other fails and so on.

That makes

$$p(success) = \sum_{i=1}^{t} f(i) = P \times \sum_{i=1}^{t} \frac{p_i}{1-p_i}$$

Let us call $\sum_{i=1}^{t} \frac{p_i}{1-p_i}$ as S

that makes

$$p(success) = P \times S$$

lets say we add one more person in the list of asking, say k

Now $P' = P(1-p_k)$ and $S' = S + \frac{p_k}{1-p_k}$

Success Probability that this person changes is

$$\Delta = P'S' - PS$$

Solving,

$$P(1-p_k).\frac{(S(1-p_k)+p_k)}{1-p_k} = P(S(1-p_k)+p_k)$$

$$\Delta = P(S - Sp_k + p_k - S) = Pp_k(1-S)$$

Now P,p_k are definitely positive but $(1-S)$ may or may not be positive.

Now, lets say we have option of adding either of two friends a or b and currently $S<1$, then,

$\Delta_a = P(1-S)p_a$

$\Delta_b = P(1-S)p_b$

We choose a if it increase more success probability(common sense) or say

$$\Delta_a > \Delta_b$$
$$\Delta_a - \Delta_b > 0$$
$$P(1-S)(p_a - p_b) > 0$$
$$(p_a - p_b) > 0$$
$$p_a > p_b$$

This means that we will select a over b only when $p_b \leq p_a$ and $S<1$

Now problem is a simple implementation and sorting one.

```
int main(){
    int n;
    cin >> n;
    V<double> arr(n);
    loop(i,0,n)cin >> arr[i];
    loop(i,0,n){
        if(arr[i] == 1){
            cout << "1.000000000000";
            return 0;
        }
    }
    sort(arr.begin(),arr.end(),greater<double>());
    double S = arr[0]/(1-arr[0]);
    double P = 1-arr[0];
    double ans = P*S;
    for(int i = 1; i < n; ++i){
        S+=(arr[i]/(1-arr[i]));
        P*=(1-arr[i]);
        ans = max(ans,P*S);
    }
    ans = max(ans,arr[0]);
    printf("%.12f",ans);
    return 0;
}
```

This PDF ends here, will add more later as I learn more stuff and solve more questions.